

Zusammenfassung

In dieser Diplomarbeit werden Konzepte zur parallelen und verteilten Simulation von HIT-Modellen entwickelt. Das *Hierarchische Tool HIT* ist ein Softwarewerkzeug zur modellbasierten Leistungsbewertung von Ereignis-diskreten Systemen. HIT-Modelle bestehen aus Komponenten, die Dienste anbieten, die wiederum von Prozessen benutzt werden. Durch die Strukturierung in Komponenten und Unterkomponenten entstehen hierarchische Modelle.

Neben analytischen und numerischen Methoden bietet HIT auch die Auswertung von Modellen durch sequentielle Simulation. Der sequentielle Simulator des HIT-Systems basiert auf *SIMULA / Class Simulation* und verwendet die *Prozeß-orientierte Sichtweise* der zu modellierenden Systeme. Ein wichtiges Ziel ist es, diese Sichtweise soweit möglich auch auf der Realisierungsebene der parallelen Simulation anzuwenden.

HIT-Modelle in einzelne Teilhierarchien zu zerlegen ist die Grundidee der Partitionierung. Komponenten bilden somit die Granularität der Zerlegung. Es wird untersucht, wie Prozesse und Komponenten interagieren und von welcher Art die ausgetauschten Nachrichten zur Ereignisvormerkung sind. Konservative und optimistische Simulationsmethoden werden auf ihre Anwendbarkeit geprüft. Jeffersons optimistisches Verfahren *Time-Warp* erweist sich dabei mit der Annullierungsstrategie *Lazy-Cancellation* als geeignet, die in der Simulation von HIT-Modellen auftretenden Sequenzen von gleichzeitigen Ereignissen mit kausalen Abhängigkeiten zu meistern.

Die Eignung der Konzepte wird durch eine prototypische Realisierung eines verteilten HIT-Simulators belegt. Schließlich wird unter Verwendung des in C++ implementierten prototypischen Simulators der Einfluß der Modellstruktur und weiterer Parameter auf die Performanz der parallelen Simulation experimentell untersucht.

Diplomarbeit:

Konzepte zur parallelen und verteilten Simulation von HIT-Modellen

Armin M. Warda

Betreuer: Prof. Dr.-Ing. H. Beilner

- Das Hierarchische Tool „HIT“
- HIT-Modelle als Systeme kommunizierender Prozesse
- Parallele / verteilte ereignisorientierte Simulation (PDES)
- Gleichzeitige Ereignisse mit kausalem Zusammenhang
- prototypische Realisierung des verteilten HIT-Simulators
- experimentelle Untersuchungen des Prototyps
- Zusammenfassung & Ausblick

HIT ist ein Software-Werkzeug zur modellbasierte Leistungsbewertung von Rechen- und Kommunikationssystemen:

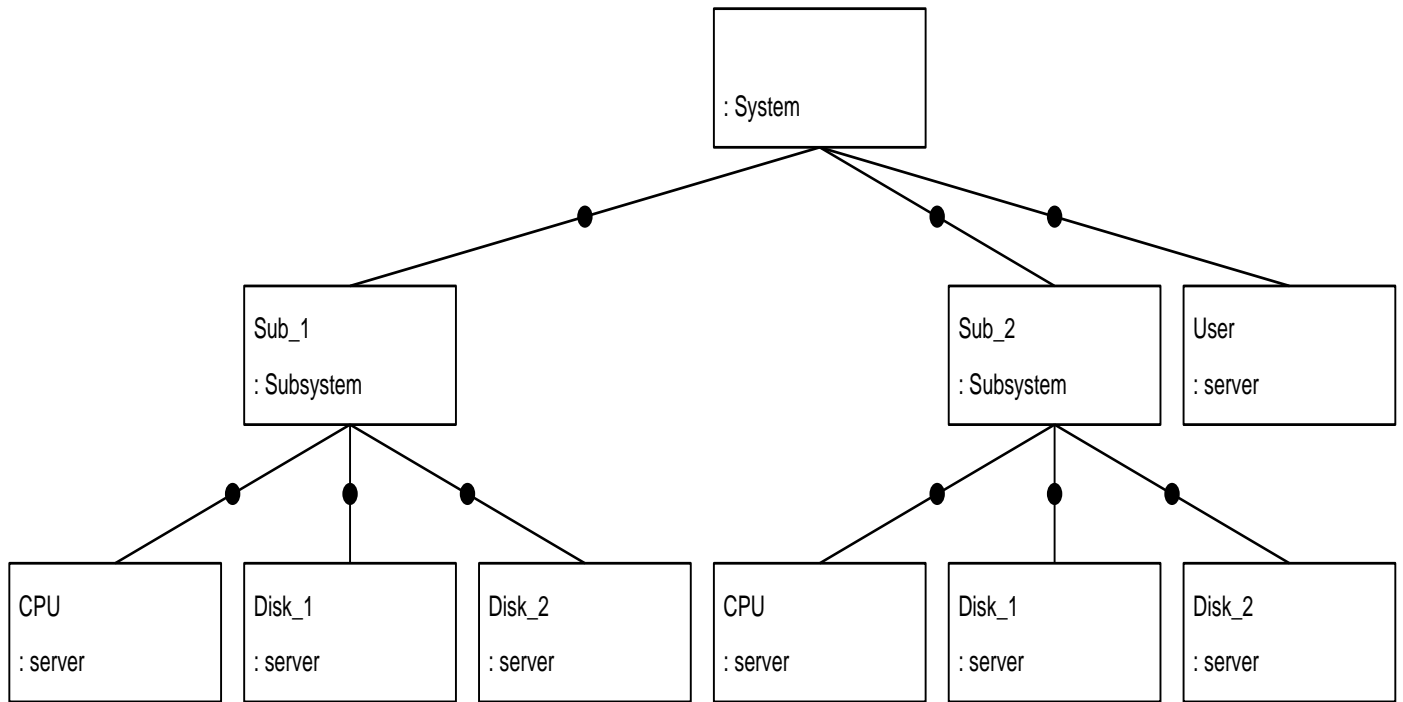
- strukturierte Modellierung
 - (vertikale) funktionale Hierarchien
 - (horizontale) Modularisierung
- Modellierungssprache HI-SLANG
- graphische Benutzeroberfläche HITGRAPHIC
- gezielten, selektive Auswertungen / Messungen von Modellen:
 - analytisch (separable bzw. Produktform-Netze)
 - numerisch (Markov-Prozesse)
 - simulativ (**sequentielle Simulation**)

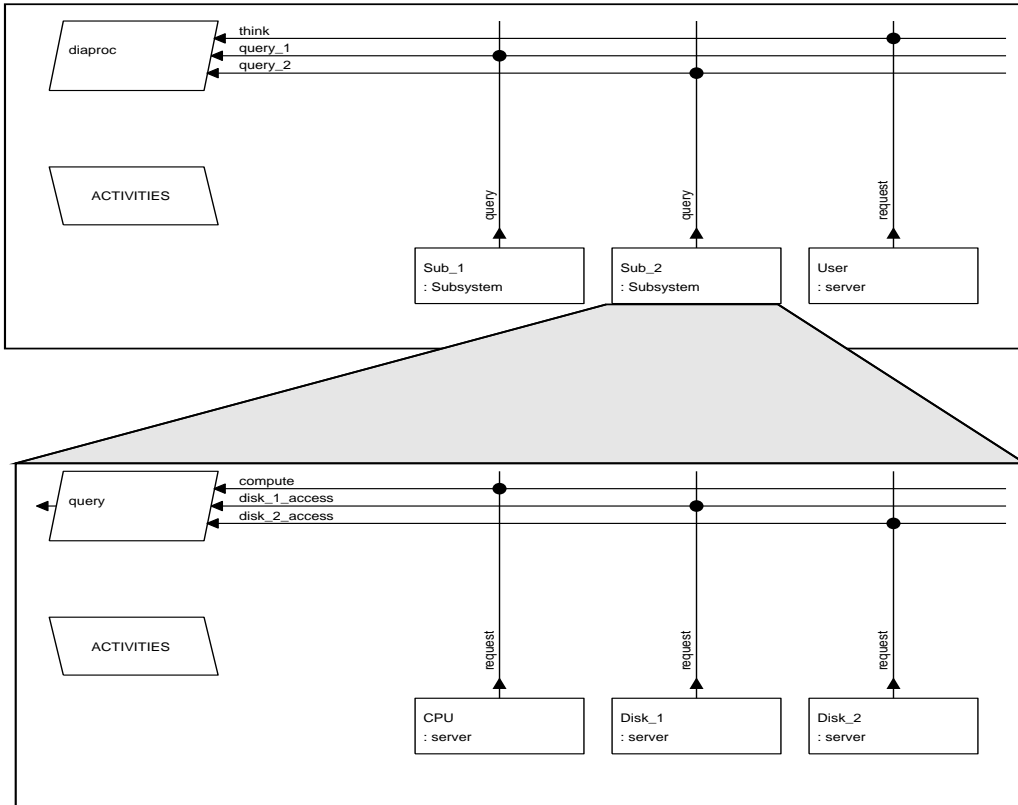
Entwickelt seit 1985 am Lehrstuhl Informatik IV der Uni. Dortmund

- Dienste (SERVICES) dienen als „Muster“ bzw. Typen für Prozesse
- Komponenten (COMPONENTS) stellen Module dar, enthalten
 - Dienste
 - Datenstrukturen
 - Prozeduren
 - **Unterkomponenten**
- vordefinierte Komponententypen:
 - Server, PrioServer, FtServer, ...
 - Semaphor, Tokenpool, Counter, ...

forcierte Sichtweise: **Last** wird auf **Maschine** aufgebracht

- Dienste einer Komponente beschreiben Lastmuster
- Unterkomponenten stellen Maschine dar





Prozesse sind die wesentlichen Akteure in einem HIT-Modell, sie . . .

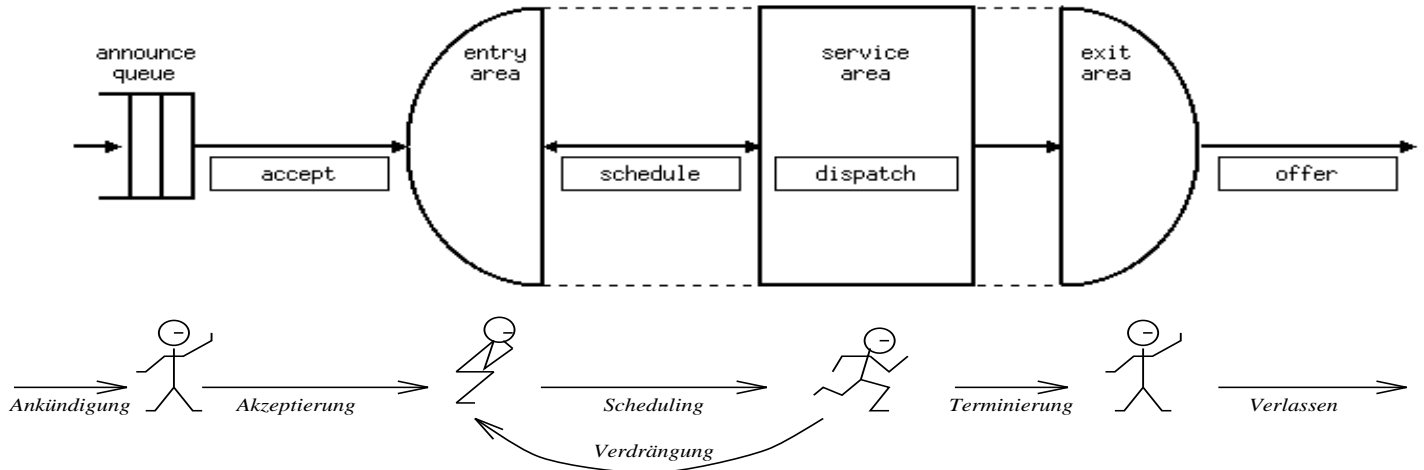
- werden durch Dienste realisiert
- werden in Komponenten lokal erzeugt
- können durch Aufrufe von benutzten Diensten, die an angebotene Dienste der Unterkomponenten gebunden sind, in Unterkomponenten „herabsteigen“

Zeitverbrauch wird realisiert durch . . .

- HOLD-Anweisung
- SPEND-Anweisung
- Dienstaufruf
- Warten auf Bedienung durch Komponente

Komponenten sind auch aktive Objekte:

- kontrollieren die in ihnen befindlichen Prozesse
- halten Prozesse in *COMPONENT AREAS*
- Kontrollprozeduren realisieren Übergänge zwischen AREAS



Vordefinierte Kontrollprozeduren:

- ACCEPT: Always, Limited, ...
- SCHEDULE: Immediate, Random, FCFS, LCFS, LCFS-PR, ...
- DISPATCH: Equal, Shared, State-Dependent, ...
- OFFER: All, ...

Aktivierungen von Kontrollprozeduren ...

- ausgelöst durch Dienstaufrufe, Prozeßterminierungen, etc.
- stellen Ereignisse dar
- können zu weiteren Aktivierungen führen
- Folgeaktivierungen können zum gleichen Modellzeitpunkt stattfinden

HIT-Modelle:

- Akteure: Komponenten und (durch Dienste realisierte) Prozesse
- Interaktion: *Shared Memory Programming Paradigm*,
wechselseitige Aktivierung (SIMULA / Class Simulation)

Komponenten ...

- sind autonome Objekte
- kapseln & kontrollieren Dienste, Datenstrukturen, Prozeduren, ...

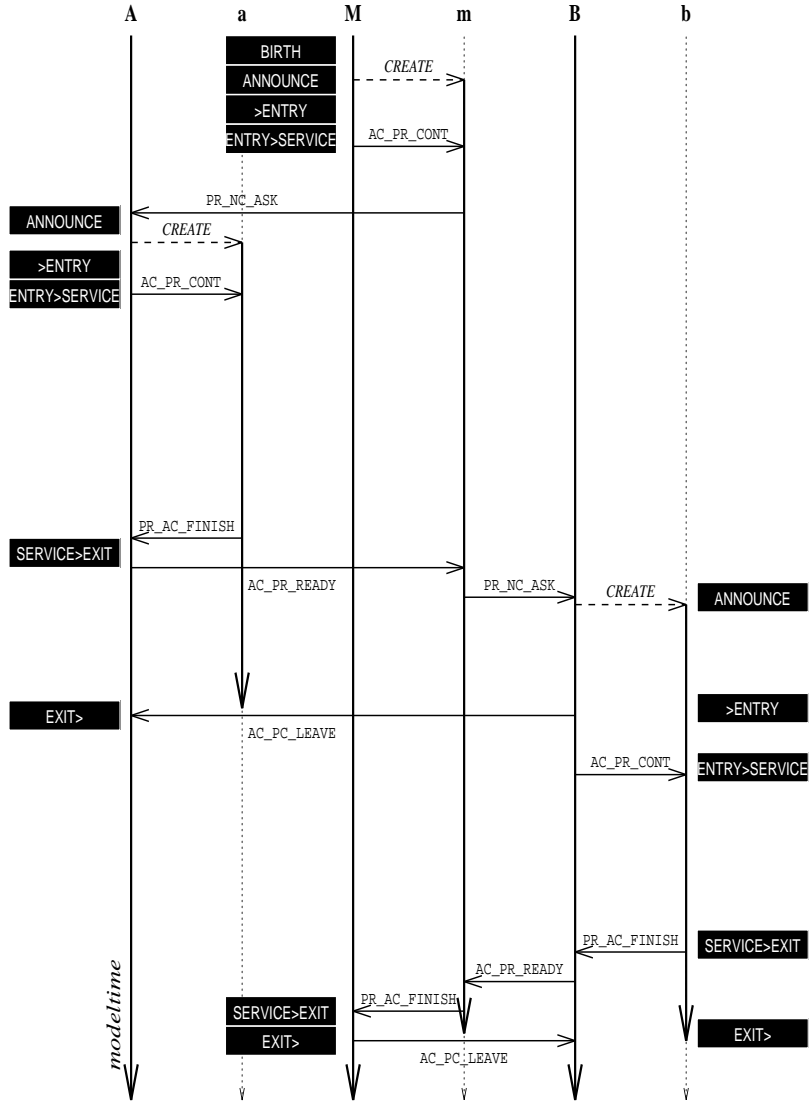
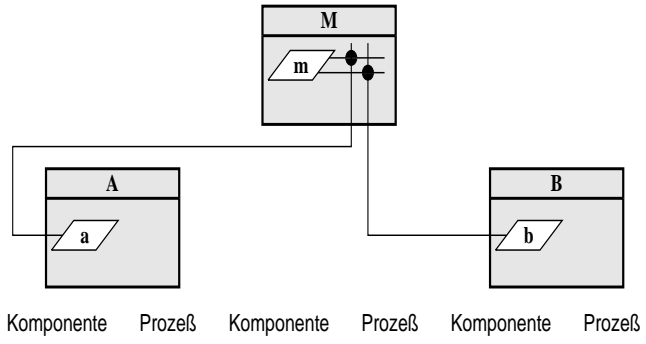
Dienste ...

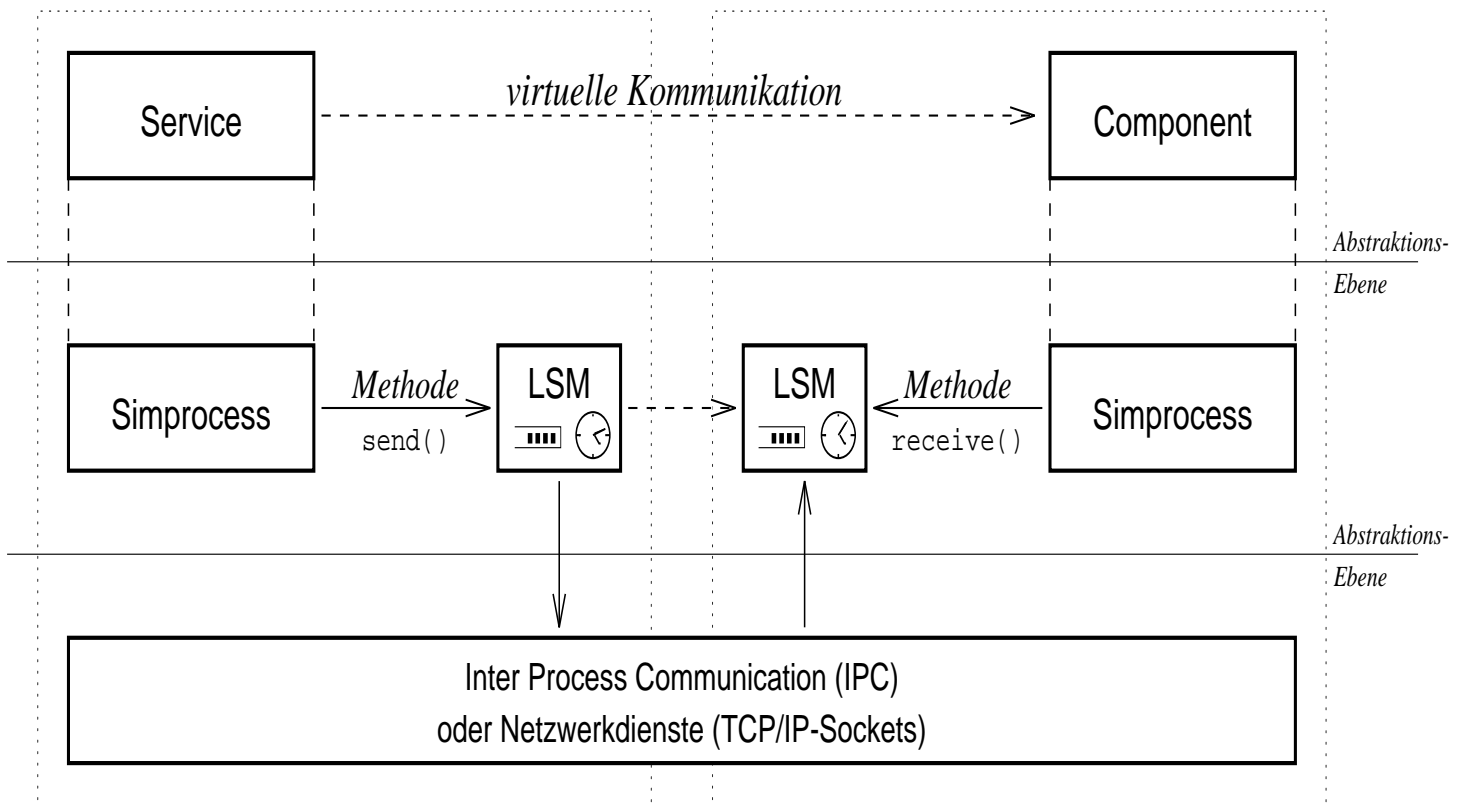
- sind wechselseitig abgekapselt
- dürfen aber auf Datenstrukturen innerhalb ihrer Komponente zugreifen
- (dürfen sogar auf Datenstrukturen außerhalb ihrer Komponente zugreifen: Blockstruktur)

Parallele / verteilte Simulation $\overset{?}{\longleftrightarrow}$ *LP-Paradigma, Message Passing*

- \implies Komponenten als Granularität der Partitionierung
- \implies innerhalb von Komponenten sequentiell simulieren

- Simulationsprozesse = Akteure =
Komponenten und (durch Dienste realisierte) Prozesse
- Ereignisvormerkung = Nachricht mit
 - Sender- / Empfängeradressen
 - Ereignistyp
 - virtueller Zeit
 - ...





Gleichzeitige Ereignisse mit kausalen Beziehungen:

1. (Folge-) Aktivierungen von Kontrollprozeduren
2. unmittelbar aufeinanderfolgende Dienstaufrufe an dieselbe Komponente
3. Dienstaufrufe ohne Modellzeitverbrauch, z.B.:
 - Tokenpool:release → immer ohne Modellzeitverbrauch
 - Tokenpool:allocate → ?

⇒ Sogar Zyklen von gleichzeitigen Ereignissen mit kausalen Beziehungen über mehrere Komponenten !

Parallele und verteilte ereignisorientierte Simulation (PDES)

LP-Paradigma: logische Prozesse (LPs) haben lokalen Zustand und kommunizieren über Nachrichten

- konservativ: **Kausalitätsverletzungen verhindern**
 - Chandy, Misra, Bryant [1979,1986]: CMB-Methode
- optimistisch: **Kausalitätsverletzungen korrigieren**
 - Jefferson [1985]: Virtual Time, Time-Warp

Konservative Methoden: **Kausalitätsverletzungen verhindern**

- Ereignisse erst simulieren, wenn alle früheren simuliert
- Austausch von Garantien
- Deadlocks müssen (z.B. durch zusätzliche Nachrichten = *Nullmessages*) verhindert werden
- oder *globale (!)* Deadlock-Detection & -Recovery

Predictability: Auf jedem Zyklus von kommunizierenden Prozessen muß ein *Lookahead* $\varepsilon > 0$ existieren: ein Prozeß kann alle im Intervall $[t, t + \varepsilon]$ von ihm zu generierende Nachrichten bestimmen, wenn er nur seinen Zustand bei t und alle bis t empfangenen Nachrichten kennt.

Zyklen von gleichzeitigen Ereignissen?

Optimistische Methoden: **Kausalitätsverletzungen korrigieren**

- Ereignisse spekulativ simulieren
- Checkpoint, Cancel, Rollback
- Annullierung kann *aggressive* oder *lazy* erfolgen

Ereignisse müssen sich innerhalb von Prozessen anhand virtueller Zeitpunkte total ordnen lassen:

virtuelle Sendezeit $T_S(e)$ = Zeitpunkt der Ereigniserzeugung
virtuelle Empfangszeit $T_R(e)$ = Zeitpunkt des Eintritts des Ereignisses

Simulation von e' generiert $e \Rightarrow T_R(e') = T_S(e) \leq T_R(e)$

Jefferson fordert hier „<“!

Gleichzeitige Ereignisse?

Was tun?

- Freiheitsgrade bei der Partitionierung einzuschränken, Gleichzeitigkeiten „Internalisieren“ ?
- HIT-Mechanismen modifizieren ?
- Konservative Methode mit globaler Deadlock-Detection & -Recovery ?
- Künstlichen Zeitfortschritt um *sehr kleines* $\varepsilon > 0$?
- Blockierung in Time-Warp ?

Definiere den **Rang** eines Ereignisses:

$$R(e) := \begin{cases} 1 & \text{falls } T_S(e) < T_R(e) \\ 1 + R(e') & \text{falls } T_S(e) = T_R(e) \end{cases}$$

wenn die Simulation von e' zur Generierung von e führt

Ersetze „Zeit“ durch „(Zeit, Rang)“, ordne diese Tupel lexikographisch

⇒ Sequenzen von gleichzeitigen Ereignissen, zwischen denen
kausale Abhängigkeiten bestehen, lassen sich total ordnen

Bemerkungen:

- „1“ willkürlich
- Ränge werden von LSMs vergeben, für Simulationsprozesse nicht sichtbar

Aggressive Cancellation: bei Rollback sofort alle voreilig produzierten Nachrichten annullieren (durch *Antimessages*)

Lazy Cancellation: Nachrichten erst dann annullieren, wenn sie sich nicht reproduzieren lassen

⇒ Sequenzen gleichzeitiger Ereignisse lassen sich mit Lazy Cancellation simulieren, wenn sie anhand von „(Zeit, Rang)“ geordnet werden

Es werden einfach durch Rollbacks so viele Anläufe genommen, bis alle Ereignisse vorliegen:

- Rollback ⇒ zurück zum Anfang der Sequenz gleichzeitiger Ereignisse
- Lazy Cancellation ⇒ keine Annullierungen für diese Sequenz
- Nachzüglerereignis in Sequenz einfügen
- neuer Anlauf

Prototypische Realisierung des verteilten HIT-Simulators

Ziel: prozeßorientierte Sichtweise des HIT-Systems (von SIMULA / Class Simulation) möglichst auch auf der Realisierungsebene

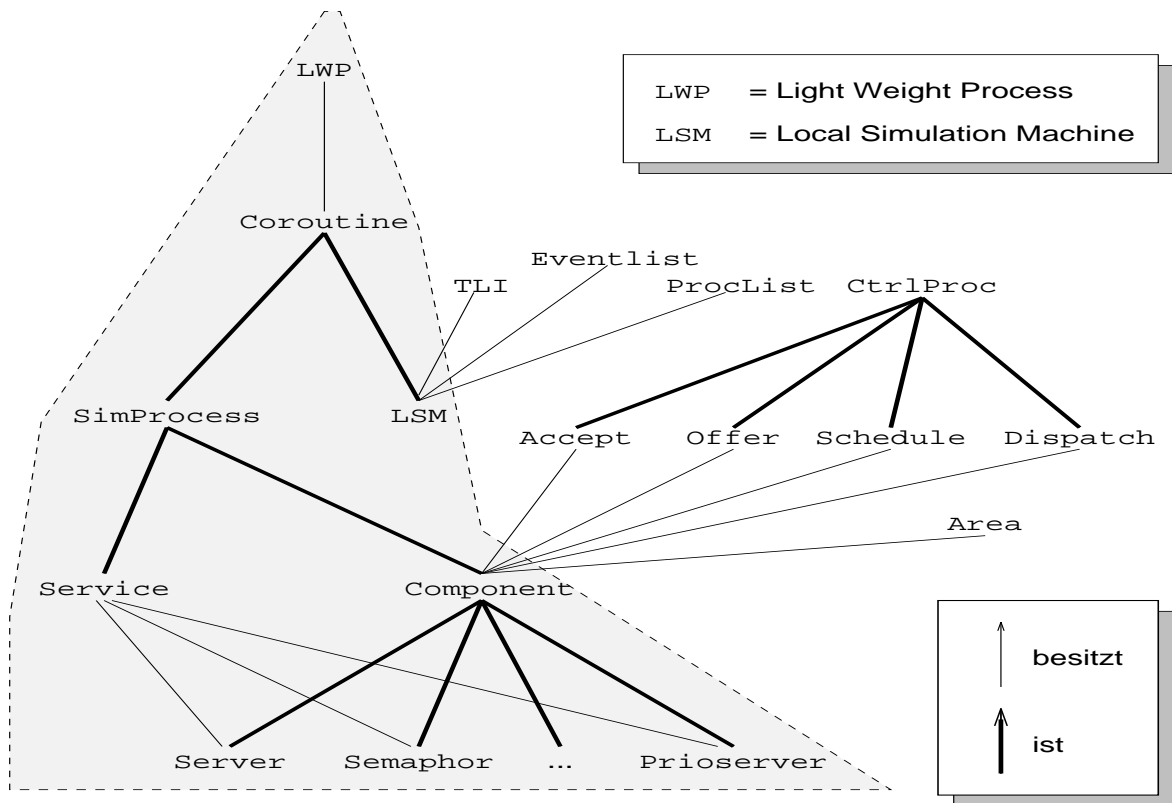
Anforderungen:

- objektorientierten Konzepte
- systemnahe Programmierung \implies C++

Transformation: HI-SLANG \longrightarrow C++ und Klassen

Einschränkung auf wesentliche Konzepte:

Komponenten (-typen), Dienste, Dienstaufrufmechanismen,
Komponentenkontrolle



Experimentelle Untersuchungen mit dem Prototypen zur verteilten Simulation von HIT-Modellen

- *Simulationsparameter:*

- Häufigkeit von

- * Checkpoints ←

- * GVT-Berechnungen

- Speichergröße

- *Modellstruktur:*

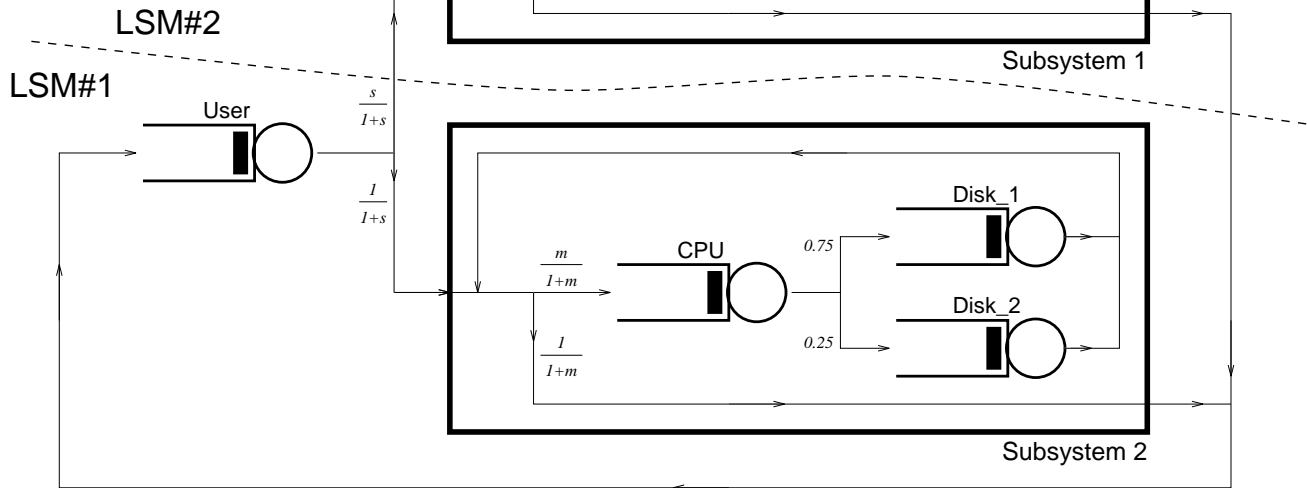
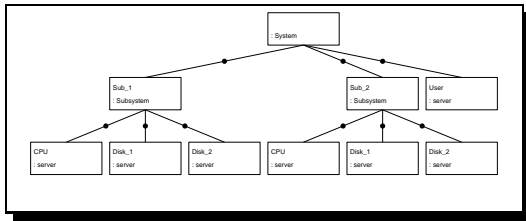
- Subsysteme

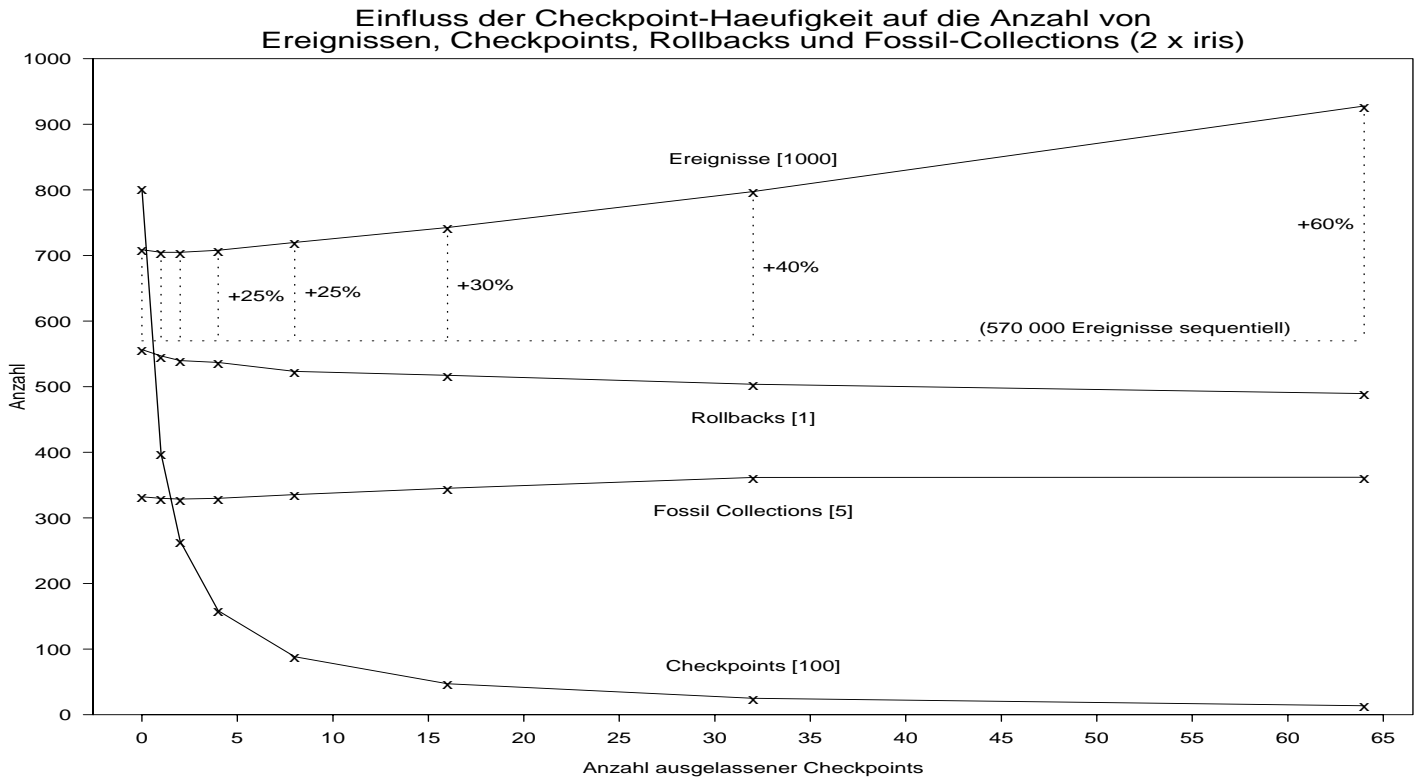
- * Partitionierung ←

- * Unabhängigkeit ←

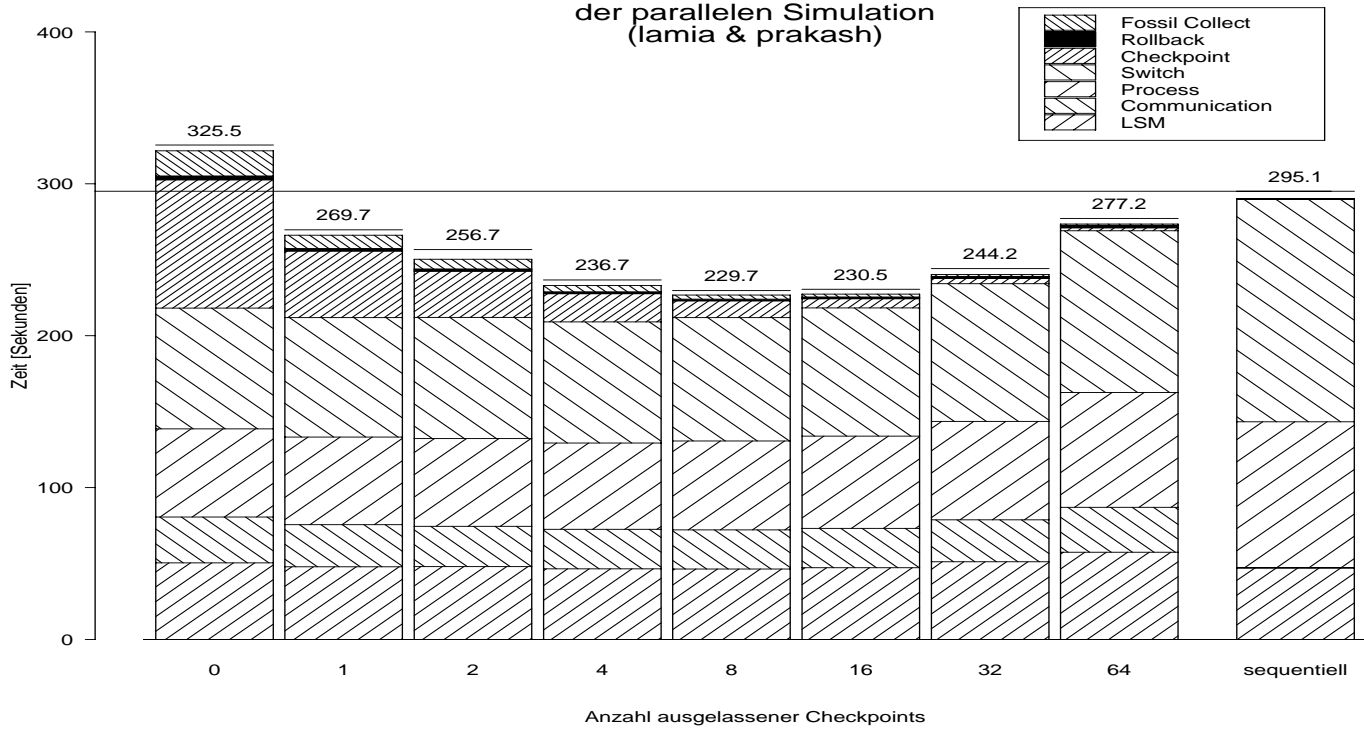
- * „Lastverteilung“

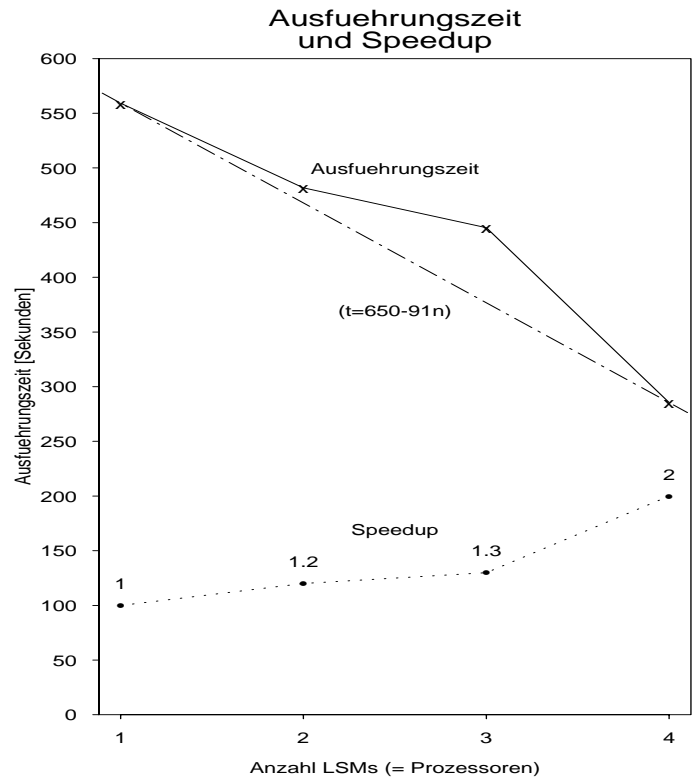
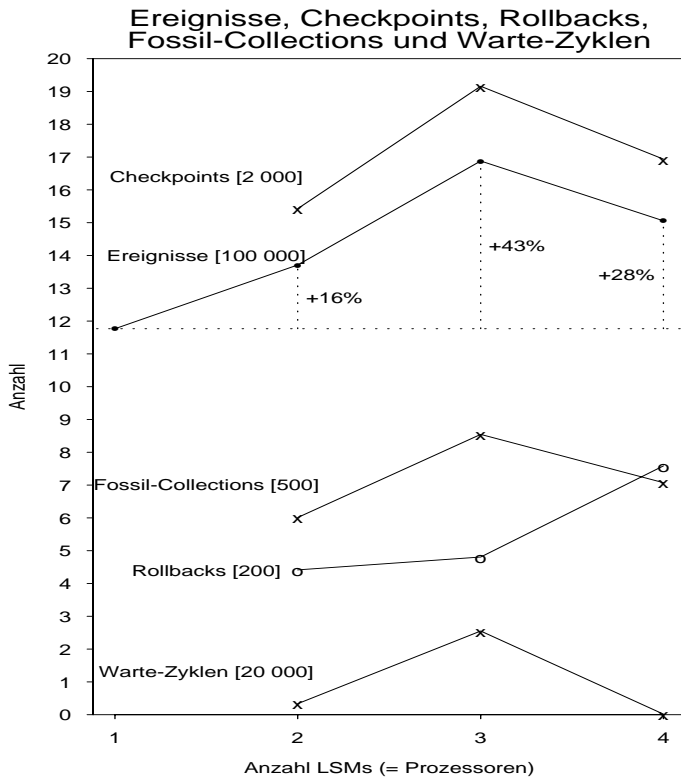
- Vereinfachung von Accept-Offer

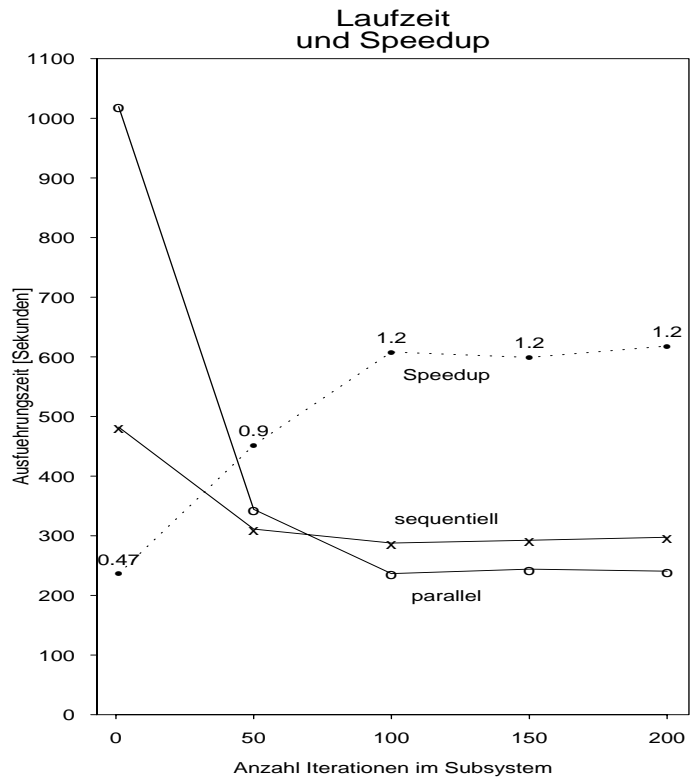
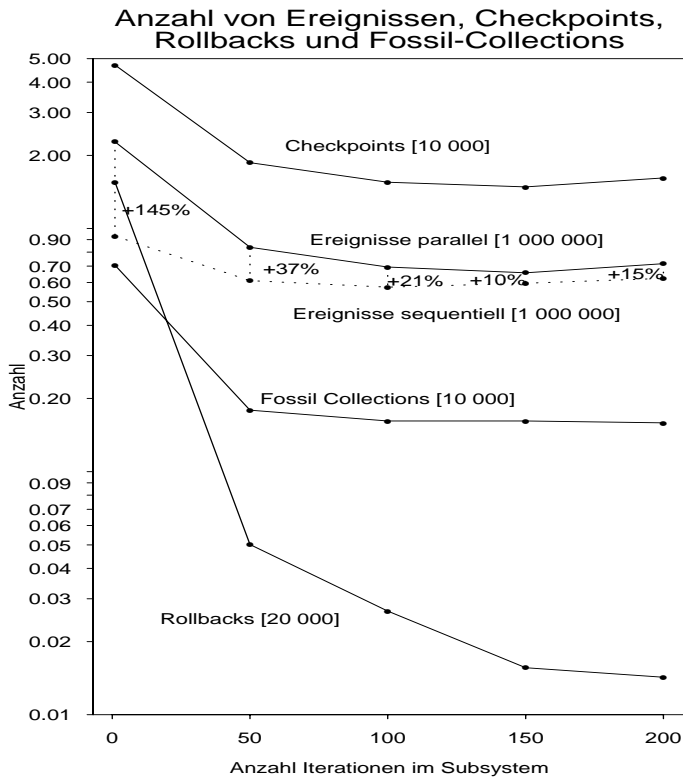




**Einfluss der Checkpoint-Haeufigkeit
auf die CPU- und Laufzeiten
der parallelen Simulation
(lamia & prakash)**







- es lohnt sich, Zustandssicherungen seltener durchzuführen
- GVT-Berechnungen nicht nur bei akutem Bedarf, sondern regelmäßig
- Partitionierung und gleichmäßige Lastverteilung wichtig
- notwendig: interne Ereignisse mind. 50 — 70 mal häufiger als Submodell-übergreifende Ereignisse
- Accept-Offer bremst

- zu wenig Speicher \Rightarrow Performanzeinbußen
- mehr Speicher bedeutet nicht unbedingt höhere Performanz

Behandelte Probleme:

- Granularität der Partitionierung: Komponenten (-hierarchien)
- HIT-Modelle als Systeme kommunizierender Prozesse
- gleichzeitige Ereignisse mit kausalem Zusammenhang
- dynamische Prozeßerzeugung und -terminierung
- transiente Fehler
- Zustandssicherung und -rücksetzung von Coroutinen

Ausgeklammerte Probleme:

- *Call-by-Reference*-Parameterübergabe
- dynamische Datenstrukturen in verteilten Systemen und im Kontext von Rollbacks

- weitere Experimente (wechselseitige Abhängigkeiten sind komplex!)
- Modelle mit Anwendungsbezug untersuchen

zunächst aber:

- „Stabilisierung“ des Prototyps
- Optimierungen hinsichtlich Performanz (Zeit & Platz)

... und ...

- größere Speedups bei komplexeren statistischen Auswertungen ?
(confidence & frequency intervals, run length control, ...)
- Vereinfachung der Komponentenkontrolle?
- Ränge zur Erfassung partieller Kausalitäten anwendbar auch auf
 - Time-Warp mit Aggressive Cancellation?
 - konservative Methoden?